A Review of Machine Learning

By: Oameed Noakoasteen

Version 03, Last Updated 2025.05

View the Presentation on Internet Archive

View the Derivation Tables on Internet Archive

View the GitHub Repository

Content

Introduction to AI & ML

- **What is Artificial Intelligence?**
- **What is Machine Learning?**
  - The Big Picture
  - A Medley of Concepts and Terms
  - Frameworks & Some Toy Datasets

Part I: Statistical Learning

- **Supervised Learning**
  - **Regression**
    - Ridge Regression, **DT[1]**
  - **Classification**
    - Logistic Regression, **DT[2]**
    - Naïve Bayes
    - Support Vector Classifier, **DT[3]**
    - k-Nearest Neighbors
    - Decision Tree
- **Unsupervised Learning**
  - **Clustering**
    - k-Means
    - Gaussian Mixture Models
  - **Dimensionality Reduction**
    - Principal Component Analysis, **DT[4]**
    - Linear Discriminant Analysis, **DT[5]**
- **Ensemble Methods, DT[6]**
- **Experiment Setup in scikit-learn**

Part II: Deep Learning

- **Feed Forward Networks, DT[7]**
- **Convolutional Layers**
- **Recurrent Layers**
- **Deep Feed Forward Networks, DT[8]**
- **Experiment Setup in TensorFlow 2**

Part III: Appendices

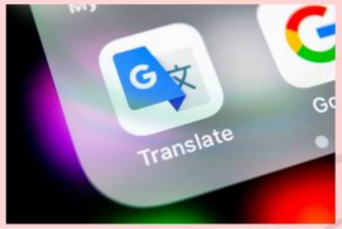
- **A Medley of Topics**
  - Bias-Variance Tradeoff, **DT[9]**
  - Bayesian Inference. **DT[10]**

What is Artificial Intelligence?

Natural Language Processing (NLP)

Speech Recognition is the task of transforming spoken sound to text; and **Text-to-Speech Synthesis** is the task of transforming text to audible sound.
Digital assistants interact with humans by listening to voice commands, executing some tasks (fetching news updates, placing purchase orders, setting timers, etc.) and then communicating the results in the form of human speech.

Machine Translation is the task of transforming text in one language into another.
A translation system translate text from more than one hundred languages.



- The question "Can Machines Think?" was answered by Alan Turing with a thought experiment that became known as the **Turing Test**. In his proposed experiment, a computer program is intelligent if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer. Turing's proposed test was an attempt at measuring intelligence by performance on some kind of open-ended behavioral task, rather than by philosophical speculation.
- Artificial Intelligence (AI) refers to computer programs that process linguistic (textual and auditory) and visual information to perceive and interact with the world like humans.

Computer Vision (CV)

Object Detection is the task of localizing (determining pixel locations using a bounding box) and classifying objects in an image.
An autonomous driving system finds the surrounding vehicles by first localizing and classifying objects in images captured by a front camera and then computing a distance estimate for each.

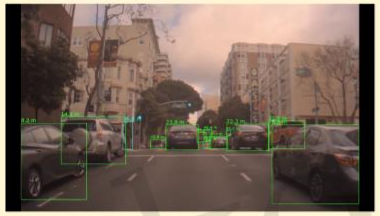


Image Transcription is the task of observing unstructured visual representation of some type of data and then transcribing the information into discrete textual form.
An image captioning system observes the scene depicted in the figure and provides a textual statement about its content.

a woman is throwing a frisbee in a park.

Pose Estimation is the task of detecting the position and orientation of objects or humans in a scene.
A robotic arm predicts 3D position and orientation of rigid objects for the purpose of manipulating them (picking, moving, and placing) in interactions with human collaborators.

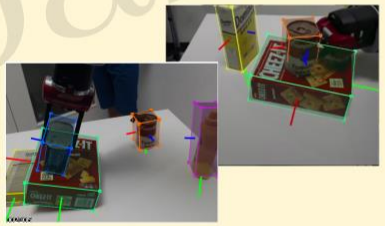


Image Super-Resolution is the task of generating a higher resolution version of the original image.
An image enhancement system enables fast acquisition of high-resolution MRI images by producing high-resolution reconstructions from low resolution acquisitions.

Image Synthesis is the task of generating a modified version of an existing image or generating an entirely new one.
A facial expression generation system generates various emotional expressions from a neutral face while preserving its identity.

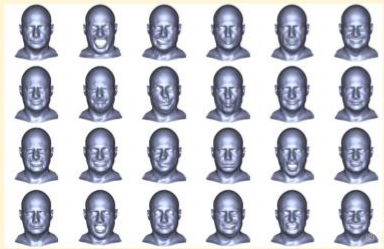


Image Denoising is the task of recovering the original image signal from its corrupted form.
An image restoration system reconstructs MRI scans from under-sampled inputs.

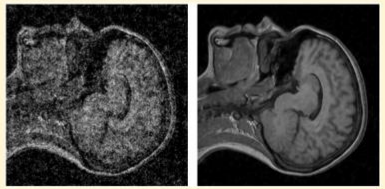


image credits: github.com/oameed/unm_phd_dissertation

What is Machine Learning?
The Big Picture

■ Machine Learning (ML) refers to computer programs that accomplish their tasks by **learning** from examples.

■ A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.
The **experience** that is available to the program is represented by the **dataset** which itself is a collection of **examples**. An example consists of a pair of **datapoint** and its corresponding **label/target**. A datapoint, is a collection of **features** that have been measured from some object or event.
For **classification** tasks the output is discrete-valued while for **regression** tasks the output is continuous-valued. Therefore, the **performance measure** is task specific. For classification tasks the **accuracy** (i.e., the proportion of examples for which the program produces the correct outputs) is used as the performance measure while for regression tasks the **average error** is used.

■ ML is about creating value from data and its recent advancements (i.e., the development of Deep Learning) have come from the pursuit of artificial intelligence.
AI can't be achieved with explicit programming because it is infeasible to manually formalize vast amounts of information in a form that computers can use. In contrast, ML offers the ability to learn a task by experiencing a collection of examples.

Training: Two Guiding Principles

The ML model, denoted by $f(\cdot, \boldsymbol{w})$, must perform its task in the **inference phase**; It will process examples that are considered to have been drawn from the data-generating distribution p_{data}, i.e., $(\boldsymbol{x}, y) \sim p_{\text{data}}$.
The ML model is constructed in the **training phase** using a collection of available examples, i.e., the **training set**, that are considered to have been drawn from the same p_{data}, i.e., $\left(\boldsymbol{x}^{\text{train}}, y^{\text{train}}\right) \sim p_{\text{data}}$. There are two guiding principles for constructing the model using the training set:

■ **Empirical Risk Minimization**

The optimal model, $f(\cdot, \boldsymbol{w}^*)$, is one that performs its task in the inference phase with minimal risk. The risk (also referred to as the **generalization error**) is defined as $R(\boldsymbol{w}) \equiv E_{(\boldsymbol{x},y)\sim p_{\text{data}}}[L(f(\boldsymbol{x}, \boldsymbol{w}), y)]$. In other words, risk is the expected value of the **error** (i.e., the output of the **loss function** L) on examples that are drawn from p_{data}. Therefore, $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\text{argmin}} \left(E_{(\boldsymbol{x},y)\sim p_{\text{data}}}[L(f(\boldsymbol{x}, \boldsymbol{w}), y)]\right)$.
However, since it is impossible to access all examples $(\boldsymbol{x}, y) \sim p_{\text{data}}$, the risk must be minimized indirectly. This is attempted by minimizing the empirical risk on the training set, hoping that doing so will also reduce the risk. The empirical risk is defined as $R_{\text{emp}}(\boldsymbol{w}) \equiv \frac{1}{m}\sum_{i=1}^{m} L\left(f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right), y_i\right)$. In other words, the empirical risk is the average error for the examples of the training set. According to the Law of Large Numbers, $R_{\text{emp}}(\boldsymbol{w})$ will represent the $R(\boldsymbol{w})$ for a large enough training set. Therefore, $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\text{argmin}} \left(\frac{1}{m}\sum_{i=1}^{m} L\left(f\left(\boldsymbol{x}^{(i)}, \boldsymbol{w}\right), y_i\right)\right)$.

■ **Likelihood Maximization**

The optimal model, $f(\cdot, \boldsymbol{w}^*)$, is one that maximizes the likelihood of the weights, or equivalently, minimizes the **negative-log-of-the-likelihood** of the weights over the entire training set. Threfore, $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\text{argmin}} \left(-\log\left[p\left(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w}\right)\right]\right)$.

Generalization

Once training is finished, the model enters the inference phase in which it will process previously unseen examples. If, after training, the gap between the average error computed on the training set and the average error computed on the **test set** (representing the generalization error) is minimal, it is said that the model **generalizes well**. Otherwise, the model **overfits** the training set. The model's generalization ability can be improved through **regularization**, which is any modification to the learning algorithm that is intended to reduce the gap between the average errors computed on training and test sets. The modified guiding principles are referred to as the **Structural Risk Minimization** and the **Posterior Maximization** respectively.

What is Machine Learning?
A Medley of Concepts and Terms

- The test set comprises examples that the system is expected to encounter in practice. Since the learning algorithm only adjusts model's parameters based on the error produced by the examples in the training set, the examples in both training and test sets must be independent and identically distributed (iid) so that minimizing the average error on the training set can affect the generalization performance of the model. In other words, the examples in the training and test sets must have been independently sampled from the same data-generating distribution p_{data}. Splitting the entire dataset into fixed training and test sets can create statistical uncertainty in estimation of the generalization error especially if the test set is small.
  The **k-fold cross-validation** is usually used to overcome these uncertainties. In this procedure, the entire dataset is split into k non-overlapping subsets and training and testing are carried out k times to obtain an estimate of the generalization error. In each trial, one of the k subsets is chosen as the test set and the rest are used as the training set. The average of the average error on the test set over the k trials is taken as the estimate of the generalization error.

- An ML model can also have non-learnable parameters, referred to as its **hyperparameters**. These are parameters that are not appropriate for the learning algorithm to adjust using the training set and must be set outside of the learning algorithm (e.g., parameters that control model's capacity, if learned on the training set, would always choose the maximum possible model capacity, resulting in overfitting).
  To choose appropriate values for hyperparameters, the training set is split into two disjoint subsets. One of these subsets is used as the training set to adjust the learnable parameters and the other, referred to as the **validation set**, is used to estimate the generalization error during training.
  The model is trained multiple times, each time with a different set of hyperparameters. After each training run, the model's performance is evaluated on the validation set. The hyperparameters that led to the best validation performance are selected. Finally, the model is retrained using the total training set with these best hyperparameters.

- A model's **capacity** controls its generalization ability. A model with low capacity will underfit while a model with high capacity will overfit. However, a model with optimal capacity does not necessarily generalize well. For example, a small training set can cause a model with optimal capacity to exhibit a large gap between its training and generalization errors.
  Since the model can be thought of as a function that maps its inputs to some outputs, the model's capacity can be thought of as its ability to fit a wide variety of functions. Therefore, as a form of regularization, the learning algorithm can be provided with a preference for one solution over another in its hypothesis space (i.e., the set of functions that the model is allowed to select as being the solution) which will result in moderation of the model's capacity.

- There is no universal or absolute best learning algorithm. The **no-free-lunch theorem** for machine learning states that, averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points. In other words, in some sense, no machine learning algorithm is universally any better than any other. To choose or design an ML algorithm, one must make assumptions about the kinds of probability distributions one encounters in real-world applications. In other words, one must first understand what kinds of data-generating distributions are relevant to the "real world" that an ML algorithm will experience, and then consider what kinds of ML algorithms will perform well on data drawn from those distributions.
  Furthermore, the no-free-lunch theorem makes it clear that there is no best form of regularization. Instead, a form of regularization must be chosen that is well suited to the task at hand.

- The **cost** refers to the quantity that the optimizer must minimize.

- The **Bayes error** is the error incurred when an ideal model makes predictions from the data-generating distribution p_{data} (an ideal model is one which simply knows p_{data}). The Bayes error defines the minimum error rate that can be achieved.
  In the case of supervised tasks, this can be due to the fact that selected features may not contain complete information about the target/label or due to fact that the system (i.e., the mapping from datapoint to target/label) is inherently stochastic.

What is Machine Learning? Frameworks & Some Toy Datasets

Statistical Learning

- **Frameworks:**
  - ***scikit-learn***
  - ***XGBoost***
- **Datasets:**
  - ***California Housing*** was derived from the 1990 U.S. census, using one row per census block group (typically 600 to 3000 people). This dataset contains 20640 examples. Each example is a pair of 8-dimensional datapoint and the target. The datapoint is constructed from the following attributes: median income in the block group, median house age in the block group, average number of rooms per household, average number of bedrooms per household, block group population, average number of household members, block group latitude, block group longitude. The target is the median house value for California districts, expressed in hundreds of thousands of dollars.
  - ***Iris Plants*** contains 150 examples in 3 classes (50 examples each). One class is linearly separable from the other two (which are NOT linearly separable from each other). Each example is a pair of 4-dimensional datapoint and the label. The datapoint is constructed from the following attributes: sepal length (cm), sepal width (cm), petal length (cm), petal width (cm). The lablels are Iris-Setosa, Iris-Versicolour and Iris-Virginica.

Deep Learning

- **Frameworks:**
  - ***TensorFlow***
  - ***PyTorch***
- **Datasets:**
  - ***MNIST*** contains 70,000 examples each of which comprises a 28x28-dimensional single-color image belonging to one of the 10 categories (i.e., digits from 0 to 9) and its corresponding label. The training/test split is 60,000/10,000.
  - ***CIFAR-10*** contains 60,000 examples each of which comprises a 32x32-dimensional colored image belonging to one of the 10 categories and its corresponding label. Each category is represented by 6,000 examples. The training/set split is 50,000/10,000.
  - ***Moving MNIST***
  - ***TED Talk Translated Transcripts***
  - ***Cora*** consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.

Statistical Learning
Supervised Learning: Regression

Ridge Regression

- **Inference**

  $\hat{y} = \tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}$

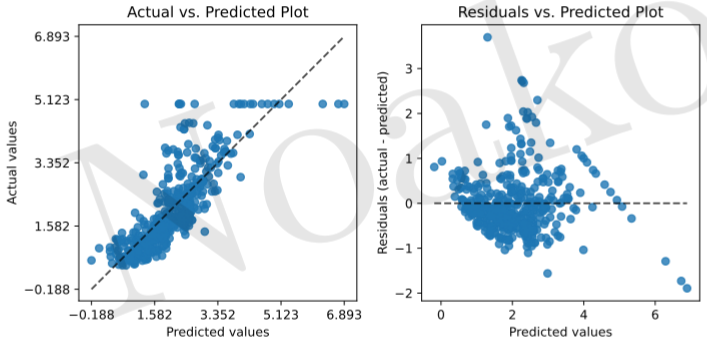
- **Training**

  $\tilde{\boldsymbol{w}}^* = \left(\tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}} + k\boldsymbol{I}\right)^{-1} \tilde{\boldsymbol{X}}^T \boldsymbol{y}$

- In the **dual formulation**, both the inference and the training operations depend only on the dot-products of datapoints. Since a **kernel** can replace the dot-product operations, the dual form enables the use of kernels which make the model non-linear.
  The dual of $\tilde{\boldsymbol{w}}^*$ is given by $\boldsymbol{\alpha} = \left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T + k\boldsymbol{I}\right)^{-1} \boldsymbol{y}$ and the single datapoint prediction is given by $\hat{y} = \boldsymbol{\alpha}^T \tilde{\boldsymbol{X}} \tilde{\boldsymbol{x}}$.

- When $k = 0$, the Ridge regression is simply referred to as the **linear regression**.
  Obtaining $\tilde{\boldsymbol{w}}^*$ using the closed form expression is referred to as the **ordinary least squares** (OLS) method while converging to it using an iterative optimization algorithm, such as **gradient descent**, is called the **least mean squares** (LMS) method.

Visualization & Quantification of the Prediction Error



Statistical Learning
Supervised Learning: Classification

Logistic Regression

Inference

$\hat{y} = 1$ if $p\left(y = 1|\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{w}}\right) \geq 0.5$; Otherwise, $\hat{y} = 0$.

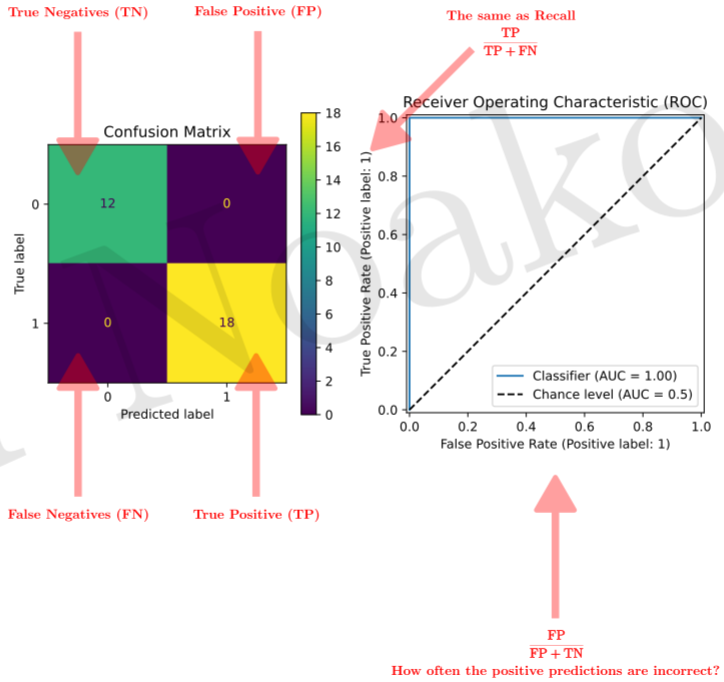
$$p\left(y = 1|\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{w}}\right) = \sigma\left(\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}\right), \ \sigma(z) \equiv \frac{1}{1 + e^{-z}}$$

Training

$$\tilde{\boldsymbol{w}}^* = \underset{\tilde{\boldsymbol{w}}}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} -\log\left[p\left(y^{(i)}|\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}^{(i)}\right)\right] + k\|\tilde{\boldsymbol{w}}\|_2^2$$

- The $\tilde{\boldsymbol{w}}^*$ has no closed form solutions and must, therefore, be obtained using the gradient descent procedure.
  The gradients of the cost with respect to the weights is given by $\frac{\partial C}{\partial \tilde{w}_j} = \frac{1}{m} \sum_{i=1}^{m} \left(\sigma\left(\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}^{(i)}\right) - y_i\right) \tilde{x}_j^{(i)}$.
  The vector containing all the gradients, denoted by $\frac{\partial C}{\partial \tilde{\boldsymbol{w}}}$, is obtained by re-writing the above expression in matrix form, i.e., $\frac{\partial C}{\partial \tilde{\boldsymbol{w}}} = \frac{1}{m} \tilde{\boldsymbol{X}}^T \left(\sigma\left(\tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}}\right) - \boldsymbol{y}\right)$.

Visualization & Quantification of the Prediction Error



- **Accuracy** answers the question: **how often the model is right?** It is defined as the ratio of the number of correct predictions to the number of total predictions, i.e., $A = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}_{f\left(\boldsymbol{x}^{(i)}\right) = y_i}\left(\boldsymbol{x}^{(i)}\right)$. **Error Rate** is the complementary concept (how often the model is wrong?) and is defined as the average of the **zero-one loss** (which is simply an indicator function), i.e., $ER = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}_{f\left(\boldsymbol{x}^{(i)}\right) \neq y_i}\left(\boldsymbol{x}^{(i)}\right)$.

- **Precision** answers the question: **how often the positive predictions are correct?**

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

- **Recall** answers the question: **how often the actual positives are identified correctly?**

$$R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

- **F-score**

$$F_1 = \frac{2}{P^{-1} + R^{-1}}$$

Statistical Learning
Supervised Learning: Classification

Support Vector Classifier

■ **Inference**

$$\hat{y} = \text{sign}\left(\boldsymbol{\alpha}^T \boldsymbol{Y}\boldsymbol{X}\boldsymbol{x} + b\right), \; y \in \{-1, +1\}$$

■ **Training**

$$\boldsymbol{\alpha}^* = \underset{\boldsymbol{\alpha}}{\text{argmin}} \; -\frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{Y}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{\alpha} + \boldsymbol{\alpha}^T\boldsymbol{1}$$

$b = \frac{1}{y} - \boldsymbol{\alpha}^{*T}\boldsymbol{Y}\boldsymbol{X}\boldsymbol{x}$ for any $(\boldsymbol{x}, y)$ on the margins.

If example $\left(\boldsymbol{x}^{(i)}, y_i\right)$ is correctly classified and is away from the margins, then $\alpha_i = 0$; If it is inside the margins or is incorrectly classified, then $\alpha_i = C^{'}$. Therefore, it follows that $0 < \alpha_i < C^{'}$ if it lies on any of the margins.

Naive Bayes

■ **Inference**

$$\hat{y} = \underset{y}{\text{argmax}} \; p\left(y|\boldsymbol{x}\right)$$

$$p\left(y = c|\boldsymbol{x}\right) \propto \left(\prod_{i=1}^{n} p\left(\boldsymbol{x}[i] = x_i|y = c\right)\right)\left(\frac{1}{m}\sum_{j=1}^{m}\mathbb{I}_c\left(y_j\right)\right)$$

$$p\left(\boldsymbol{x}[i] = x_i|y = c\right) = \frac{\sum_{j=1}^{m}\mathbb{I}_c(y_j)\mathbb{I}_{x_i}\left(\boldsymbol{x}^{(j)}[i]\right)}{\sum_{j=1}^{m}\mathbb{I}_c(y_j)}, \text{ for } \textbf{Categorical Features}$$

$$p\left(\boldsymbol{x}[i] = x_i|y = c\right) = \mathcal{N}\left(\mu_{ic}, \sigma_{ic}^2\right), \text{ for } \textbf{Continuous Features}$$

k-Nearest Neighbors

■ **Inference**

$\hat{y}$ is the majority label of the k nearest neighbors of $\boldsymbol{x}$.

■ The underlying assumption is that datapoints that are close to each other have similar labels. The **closeness** is determined by a **distance metric** such as Euclidean or Minkowski.

Decision Tree

■ **Inference**

$\hat{y}$ is the **leaf** label of $\boldsymbol{x}$. The leaf of $\boldsymbol{x}$ is determined by traversing a tree structure defined by thresholds on some of the features of $\boldsymbol{x}$.

■ **Training**

1. Repeat recursively until there are no more features left to work with:
   (a). Divide the entire dataset into two subsets that are purer in terms of their class membership using a threshold on a certain feature. All features and all possible thresholds are considered, and an impurity cost is computed for each. That feature and threshold combination is chosen which minimizes the impurity cost (e.g., entropy or Gini impurity).
   (b). For each newly created subset, if all its datapoints have the same label, it will not be divided any further and a leaf (with that label) is associated with it. Otherwise, it will be further divided using the same procedure.
2. At the end, to each impure subset is associated a leaf which is labeled with the majority label in that subset.

■ The underlying assumption is that datapoints that are close to each other have similar labels. The **closeness** is determined by Membership in the same leaf.

Statistical Learning
Unsupervised Learning: Clustering

k-Means

- 1. Initialize cluster centroids $\boldsymbol{\mu}^{(j)}$ (for $j = 1, \cdots, k$) with random values.
  2. Repeat until convergence (i.e., until assignments $\boldsymbol{r}^{(i)}$ don't change):
     (a). E step: compute the assignments

$$r_{j^*}^{(i)} = 1, \; j^* = \underset{j}{\operatorname{argmin}} \|\boldsymbol{\mu}^{(j)} - \boldsymbol{x}^{(i)}\|_2^2$$

     (b). M step: update cluster centroids using the newly computed assignments

$$\boldsymbol{\mu}^{(j)} = \frac{1}{\sum_{l=1}^{m} r_j^{(l)}} \sum_{i=1}^{m} r_j^{(i)} \boldsymbol{x}^{(i)}$$

- To each datapoint $\boldsymbol{x}_{n\times 1}$ an assignment vector $\boldsymbol{r}_{k\times 1}$ is associated such that $r_j \in \{0,1\}$ and $\sum_{j=1}^{k} r_j = 1$. The cost is defined as $C = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} r_j^{(i)} \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)}\|_2^2$.

Gaussian Mixture Model

- 1. Initialize parameters π_j, $\boldsymbol{\mu}^{(j)}$, $\boldsymbol{\Sigma}^{(j)}$ (for $j = 1, \cdots, k$) with random values.
  2. Repeat until convergence (i.e., until the change in parameters π_j, $\boldsymbol{\mu}^{(j)}$, $\boldsymbol{\Sigma}^{(j)}$, or alternatively in the log-likelihood, falls below some threshold):
     (a). E step: compute the responsibilities

$$p\left(z_j^{(i)} = 1 | \boldsymbol{x}^{(i)}\right) = \frac{\pi_j}{\sum_{l=1}^{k} \pi_l N\left(\boldsymbol{x}^{(i)} | \boldsymbol{\mu}^{(l)}, \boldsymbol{\Sigma}^{(l)}\right)} N\left(\boldsymbol{x}^{(i)} | \boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)}\right)$$

     (b). M step: update parameters using the newly computed responsibilities

$$\pi_j = \frac{1}{m}\sum_{i=1}^{m} p\left(z_j^{(i)} = 1 | \boldsymbol{x}^{i}\right)$$

$$\boldsymbol{\mu}^{(j)} = \frac{1}{m}\sum_{i=1}^{m} \frac{p\left(z_j^{(i)} = 1 | \boldsymbol{x}^{(i)}\right)}{\pi_j} \boldsymbol{x}^{(i)}$$

$$\boldsymbol{\Sigma}^{(j)} = \frac{1}{m}\sum_{i=1}^{m} \frac{p\left(z_j^{(i)} = 1 | \boldsymbol{x}^{(i)}\right)}{\pi_j} \left(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)}\right)\left(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)}\right)^T$$

- For each datapoint $\boldsymbol{x}_{n\times 1}$ the $p(\boldsymbol{x})$ is modeled by associating with it a $\boldsymbol{z}_{k\times 1}$ and then computing the marginal distribution of $\boldsymbol{x}$ from the joint distribution $p(\boldsymbol{x}, \boldsymbol{z})$, i.e., $p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{z})\, p(\boldsymbol{x}|\boldsymbol{z})$. The $\boldsymbol{z}$ is defined such that $z_j \in \{0,1\}$ and $\sum_{j=1}^{k} z_j = 1$. With this definition, $p(\boldsymbol{z}) = \prod_{j=1}^{k} \pi_j{}^{z_j}$ in which $\pi_j \equiv p(z_j = 1)$. Furthermore, $p(\boldsymbol{x}|\boldsymbol{z}) = \prod_{j=1}^{k} N\left(\boldsymbol{x}|\boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)}\right)^{z_j}$. Therefore, $p(\boldsymbol{x}) = \sum_{j=1}^{k} \pi_j N\left(\boldsymbol{x}|\boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)}\right)$ and the likelihood is given by $p(\boldsymbol{X}) = \prod_{i=1}^{m}\sum_{j=1}^{k} \pi_j N\left(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)}\right)$.

Statistical Learning
Unsupervised Learning: Dimensionality Reduction

The Curse of Dimensionality

As the number of dimensions increases, the distance between datapoints loses its significance as a measure of closeness. A simple experiment can illustrate this phenomenon: (a) define space S as the n-dimensional hyper-cube $[0,1]^n$. (b) select a datapoint D_0 and define "closeness" as being located within a fixed volume V_0 containing this datapoint; This volume can be defined as a hyper-cube with edge length l_0, i.e., $V_0 = l_0^n$. (c) uniformly sample m datapoints in S (d) the k datapoints that have the smallest distances to D_0 are located within a volume defined by a hyper-cube with edge length $l = \left(\frac{k}{m}\right)^{\frac{1}{n}}$. From this setting, it can be observed that as the number of dimensions increase (1) the k datapoints that have the smallest distances to D_0 are located within an increasingly larger volume (2) the number of sampled datapoints must increase exponentially to have the v smallest-distance datapoints within the fixed volume V_0.

Principle Component Analysis

- $\boldsymbol{z}_{k\times 1} = \boldsymbol{W}^T_{n\times k}\boldsymbol{x}_{n\times 1}$
- $\boldsymbol{W}^* \equiv \left[\boldsymbol{w}^{(1)}_{n\times 1} \cdots \boldsymbol{w}^{(k)}_{n\times 1}\right]$ is constructed from the k eigen-vectors of $\boldsymbol{X}^T\boldsymbol{X}$ corresponding to its k largest eigen-values.

Linear Discriminant Analysis

- $z = \boldsymbol{w}^T_{n\times 1}\boldsymbol{x}_{n\times 1}$
- $\boldsymbol{w}^*$ is the eigen-vector of $\boldsymbol{S}_B^{-1}\boldsymbol{S}_W$ corresponding to its largest eigen-value.
  $\boldsymbol{S}_B \equiv (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^T$ in which $\boldsymbol{\mu}_c$ is the mean of the datapoints in class c, i.e., $\boldsymbol{\mu}_c = \frac{1}{m_c}\boldsymbol{X}^T_{m_c\times n}\mathbf{1}$.
  $\boldsymbol{S}_W \equiv \boldsymbol{\Sigma}_0 + \boldsymbol{\Sigma}_1$ in which $\boldsymbol{\Sigma}_c$ is the Covariance Matrix of the datapoints in class c.

Statistical Learning
Ensemble Methods

Bootstrap Aggregation (Bagging)

■ **Inference**

The ensemble's prediction is the aggregated predictions from all the weak learners.
For regression, aggregation is achieved by **averaging**, i.e., $F(\boldsymbol{x}) = \frac{1}{M}\sum_{j=1}^{M} f_j(\boldsymbol{x})$.
For classification, it is achieved by selecting the **majority label**, i.e., $F(\boldsymbol{x}) = \text{mode}(f_1(\boldsymbol{x}), \cdots, f_M(\boldsymbol{x}))$.

■ **Training**

1. Generate M bootstrap sets from the training set.
2. Train a separate model (also referred to as **weak learner**) on each bootstrap set.

■ A bootstrap set is obtained from an original set (of size m) by sampling the original m times with replacement.
Any example from the original set that is not present in the bootstrap set is referred to as an **out-of-bag** example.

■ The **Random Forest** is an example of bagging in which the Decision Tree is used as the weak learner.

Gradient Boosting

■ **Inference**

The ensemble's prediction is given by $F(\boldsymbol{x}) = \text{sign}\left(\sum_j \alpha_j f_j(\boldsymbol{x})\right)$ in which $F(\boldsymbol{x}), f(\boldsymbol{x}) \in \{+1, -1\}$.

■ **Training: AdaBoost**

1. Initiallize $\boldsymbol{w}^{(1)} = \frac{1}{m}\mathbf{1}_{m\times 1}$.
2. For $j = 1, \cdots, M$:
   (a). Fit classifier f_j (also referred to as **weak learner**) by minimizing the sum of **weighed zero-one loss** over the training set, i.e., $\epsilon_j \equiv \sum_{i=1}^{m} w_i \mathbb{I}_{f(\boldsymbol{x}^{(i)})\neq y_i}\left(\boldsymbol{x}^{(i)}\right)$.
   (b). Compute ϵ_j and the step size, i.e., $\alpha_j = \frac{1}{2}\ln\left(\frac{1-\epsilon_j}{\epsilon_j}\right)$.
   (c). If $\epsilon_j < 0.5$ (otherwise, exit the loop):
   Update the weights using the transformation $w_i^{(j+1)} = \left[\frac{1}{2\sqrt{\epsilon_j(1-\epsilon_j)}} e^{-\alpha_j y_i f_j\left(\boldsymbol{x}^{(i)}\right)}\right] w_i^{(j)}$.

■ The next weak learner to be added to the ensemble must satisfy $f^* = \underset{f}{\text{argmin}} \sum_{i=1}^{m} \frac{\partial C}{\partial F\left(\boldsymbol{x}^{(i)}\right)} f\left(\boldsymbol{x}^{(i)}\right)$ in which $\sum_{i=1}^{m} \frac{\partial C}{\partial F\left(\boldsymbol{x}^{(i)}\right)} f\left(\boldsymbol{x}^{(i)}\right) < 0$. In the AdaBoost algorithm, the loss function used is exponential, i.e., $L(F(\boldsymbol{x}), y) \equiv e^{-yF(\boldsymbol{x})}$ resulting in $C(F) = \frac{1}{m}\sum_{i=1}^{m} e^{-y_i F\left(\boldsymbol{x}^{(i)}\right)}$.

■ As iterations progress, the weighting coefficients are increased for data points that are misclassified and decreased for data points that are correctly classified.
Successive classifiers are therefore forced to place greater emphasis on points that have been misclassified by previous ones.

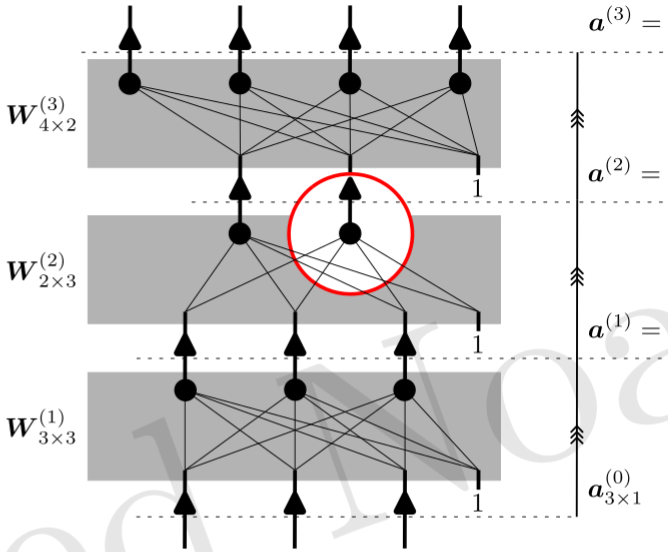
Statistical Learning
Experiment Setup in scikit-learn

```
...

def visualize(paths,y,y_hat):
  ...

def main():
  from sklearn import datasets
  from sklearn import preprocessing
  from sklearn import model_selection
  from sklearn import linear_model

  ...

  data  = datasets.<method-for-fetching-the-dataset>
  data  = model_selection.train_test_split (data["data"  ],
                                            data["target"],
                                            ...
                                                          )
  X,y   = data[0], data[2]
  X     = preprocessing.scale(X)

  model = linear_model.<the-model>
  model.fit(X,y)

  X,y   = data[1], data[3]
  X     = preprocessing.scale(X)

  y_hat = model.predict(X)

  visualize(paths,y,y_hat)

if __name__=="__main__":
  main()
```

Deep Learning
Feed-Forward Neural Networks (FFNNs)

- **Artificial Neural Networks** (ANNs) are a category of ML models. They are computational models that are composed of multiple processing layers intended to learn representations of data at multiple levels of abstraction.
- Each **layer** consists of several **units** (also referred to as **neurons**) that independently process that layer's input vector. A unit produces its output by multiplying each element of the input vector by a dedicated **weight**, summing them together along with a **bias** and, finally, applying a **non-linear function** to the total sum.
  Since each unit processes all the elements of the input, this type of layer is referred to as a **Fully-Connected** layer.
- The output of a layer, which is the vector containing the outputs from all the units in that layer, is computed as the multiplication of the **expanded weight matrix** and the **expanded input vector**. The output of each layer becomes the input to the next layer and the computations continue.
- In the training phase, the learning algorithm will need to compute the gradient of the cost w.r.t the weights and biases. For this purpose, the $\boldsymbol{\delta}$ vector at the input of each layer is computed as the multiplication of the weight matrix with the scaled $\boldsymbol{\delta}$ vector at the output of that layer.

Forward Pass



$$\boldsymbol{a}^{(3)} = f\left(\tilde{\boldsymbol{W}}^{(3)}\tilde{\boldsymbol{a}}^{(2)}\right)$$

$$\boldsymbol{a}^{(2)} = f\left(\tilde{\boldsymbol{W}}^{(2)}\tilde{\boldsymbol{a}}^{(1)}\right)$$

$$\boldsymbol{a}^{(1)} = f\left(\tilde{\boldsymbol{W}}^{(1)}\tilde{\boldsymbol{a}}^{(0)}\right)$$

$$\boldsymbol{a}^{(0)}_{3\times1}$$

Back Propagation

$$\boldsymbol{\delta}^{(3)}_{4\times1}$$

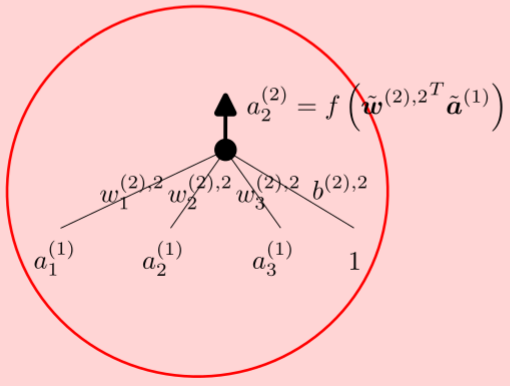
$$\boldsymbol{\delta}^{(2)} = \boldsymbol{W}^{(3)^T}\left(\boldsymbol{f}'^{(3)} \odot \boldsymbol{\delta}^{(3)}\right)$$

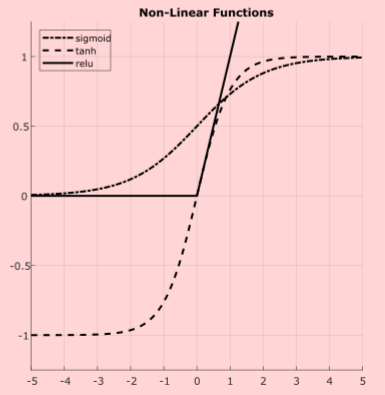
$$\boldsymbol{\delta}^{(1)} = \boldsymbol{W}^{(2)^T}\left(\boldsymbol{f}'^{(2)} \odot \boldsymbol{\delta}^{(2)}\right)$$

$$\frac{\partial C}{\partial \tilde{\boldsymbol{W}}^{(3)}} = \left(\boldsymbol{f}'^{(3)} \odot \boldsymbol{\delta}^{(3)}\right)\tilde{\boldsymbol{a}}^{(2)^T}$$

$$\frac{\partial C}{\partial \tilde{\boldsymbol{W}}^{(2)}} = \left(\boldsymbol{f}'^{(2)} \odot \boldsymbol{\delta}^{(2)}\right)\tilde{\boldsymbol{a}}^{(1)^T}$$

$$\frac{\partial C}{\partial \tilde{\boldsymbol{W}}^{(1)}} = \left(\boldsymbol{f}'^{(1)} \odot \boldsymbol{\delta}^{(1)}\right)\tilde{\boldsymbol{a}}^{(0)^T}$$





Deep Learning
Convolutional Layers

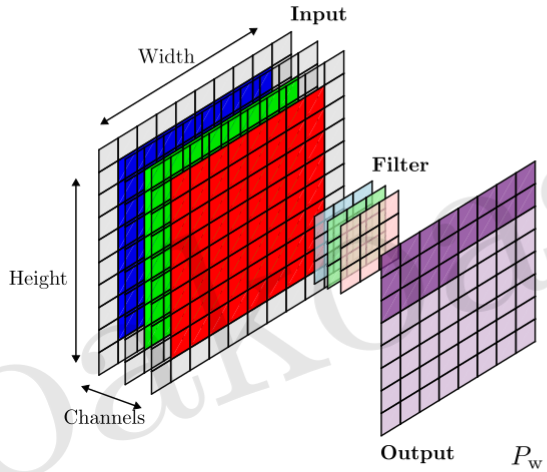
- A convolutional layer is a layer which is specialized for processing inputs that are in the form of a grid of values.
- This type of layer can learn a single model for processing local neighborhoods of all the positions across the input grid.
  This capability results from the layer's **sparse interactions** with its inputs and its output's **equivariance to translations**. The first property is achieved by restricting the interactions of the layer's set of weights (which are much smaller in size compared to the input) to only a local region of the input. The second property is achieved by **weight sharing** between different local regions.
- Each element of the output is produced by sliding the set of weights, i.e., the **filter**, across the input. For each position, the weights in each channel of the filter are multiplied by their corresponding elements in the input (i.e., same channel and same position in the input) and the results are summed together. The sums from all the channels are added together along with a bias. Finally, a non-linear function is applied to the total sum.

- $$W_{\text{out}} = \frac{W_{\text{in}} + 2P_{\text{w}} - D_{\text{w}}(F_{\text{w}} - 1) - 1}{S_{\text{w}}} + 1$$

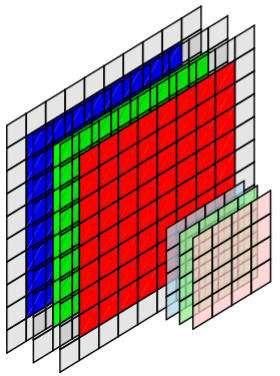
  $$H_{\text{out}} = \frac{H_{\text{in}} + 2P_{\text{h}} - D_{\text{h}}(F_{\text{h}} - 1) - 1}{S_{\text{h}}} + 1$$

Padding refers to insertion of zeros to both sides (beginning and end) of the input in each dimension.
Stride refers to the distance between two consecutive positions of the filter.
Dilation refers to the inflation of the filter by inserting spaces between filter elements; This provides a convenient way to increase the receptive field (i.e., the local region of interaction) of the filter without increasing its size.

Same/Valid Convolution

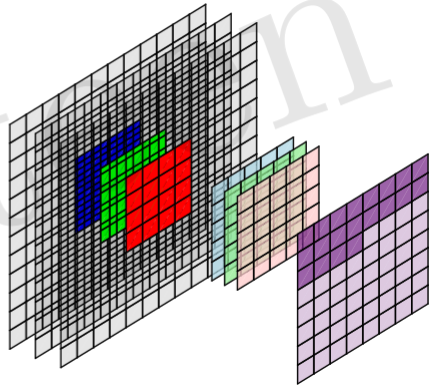


$P_{\text{w}} = P_{\text{h}} = 1$
$S_{\text{w}} = S_{\text{h}} = 1$
$D_{\text{w}} = D_{\text{h}} = 1$



$P_{\text{w}} = P_{\text{h}} = 1$
$S_{\text{w}} = S_{\text{h}} = 2$
$D_{\text{w}} = D_{\text{h}} = 1$

Transpose Convolution

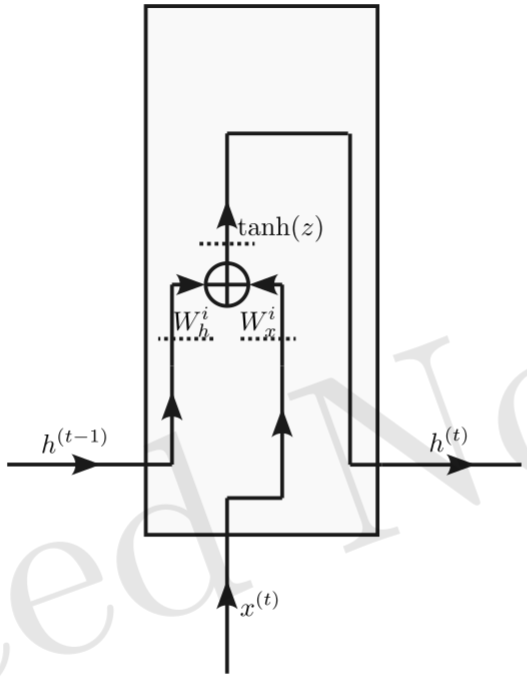


$P_{\text{w}} = P_{\text{h}} = 4$
$S_{\text{w}} = S_{\text{h}} = 1$
$D_{\text{w}} = D_{\text{h}} = 1$

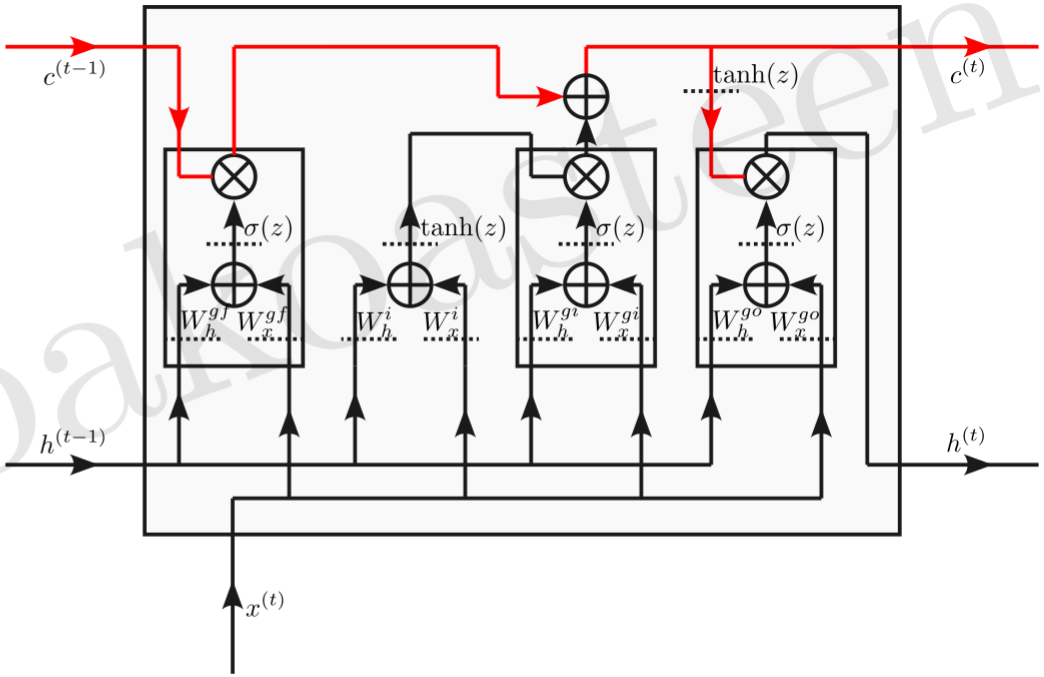
Deep Learning
Recurrent Layers

- A recurrent layer is a layer which is specialized for processing inputs that are in the form of sequences.
- This type of layer can learn a single model for processing all the positions of an input sequence. This capability results from **weight sharing** across all the positions of the input sequence which is achieved through **recurrent updates**.
  A recurrent update refers to the application of the same forward propagation operation (i.e., multiplication by the same set of weights) to produce the current position in the output from the current input and the previous position in the output.
- The LSTM is the most suitable architecture for learning **long-term dependencies**. It comprises a **Constant Error Carousel** (CEC) and three gates, namely, the **forget gate**, the **input gate** and the **output gate**. The CEC acts as a memory mechanism which, selectively, holds the value from the previous input.
  As such, the CEC provides an unattenuated/unmagnified path for the gradients of the cost as they propagate backwards through time and, therefore, helps to mitigate the **vanishing/exploding gradients** problem in long sequences.

Vanilla RNN



LSTM



Deep Learning
Deep Feed Forward Networks

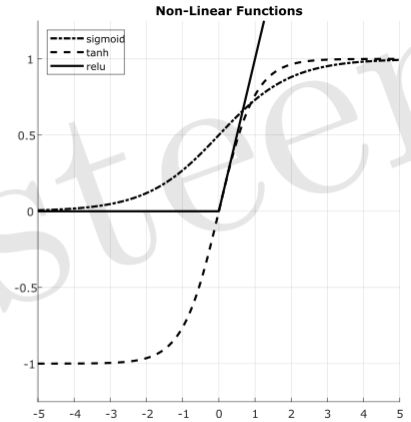
- The **Universal Approximation Theorem** (UAT) states that a FFN with a linear output layer and at least one hidden layer which has any non-linear activation function (i.e., either a ReLU or a 'squashing' function such as the sigmoid) can approximate any Borel-measurable function (i.e., a continuous function defined on a closed and bounded subset of $\mathbb{R}^n$) with any desired non-zero amount of error, provided that the FFN is given enough hidden units.
  - While the UAT guarantees the existence of the function that would generalize to unseen examples, it provides no guarantees that the learning algorithm will be able to learn that function. Specifically, learning can fail either due to the optimization algorithm's inability to find the set of parameters that corresponds to the desired function or due to overfitting.
  - While the UAT guarantees the existence of a FFN large enough to achieve any desired degree of accuracy, it provides no indication as to how large it should be.
  - In summary, a FFN with a single hidden layer is sufficient to represent any function, but the layer may fail to learn at all or to generalize and, also, it may be infeasibly large.
- FFNs are structured as deep stacks of layers, not just to alleviate the problems that the shallow networks present, but also to arrange for the sought-after function to be constructed by composition of several simpler ones.
- Training deep FFNs is fraught with difficulties. A major obstacle for the learning algorithm is the failure of the back-propagation procedure. The main failure mechanisms are the **vanishing/exploding gradients** and the **shattered gradients**.

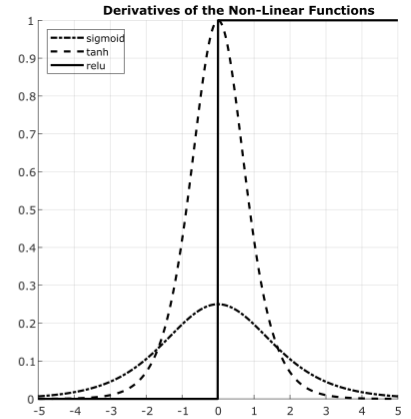
Vanishing/Exploding Gradients

When the gradients of the cost w.r.t. the weights and biases of the l^{th} layer, i.e., $\frac{\partial C}{\partial \tilde{\boldsymbol{W}}^{(l)}}$, become so small that they can't effect meaningful change in that layer's parameters, the learning algorithm is said to suffer from **vanishing gradients**.
Likewise, when $\frac{\partial C}{\partial \tilde{\boldsymbol{W}}^{(l)}}$ become so large that they cause unstable change, the learning algorithm is said to suffer from **exploding gradients**.

- **Non-saturating Non-linear Functions: ReLU**

  In computation of $\boldsymbol{\delta}$ for deeper (i.e., lower) layers, the derivative of the non-linear function from all the previous layers are multiplied together. Since for the ReLU $f'(z \geq 0) = 1$, these multiplications will not attenuate the elements of the $\boldsymbol{\delta}$.
  While $f'(z < 0) = 0$ does contribute to zero elements in $\boldsymbol{\delta}$ and, consequently, in $\frac{\partial C}{\partial \tilde{\boldsymbol{W}}}$, the back-propagation procedure will work as long as the gradients can propagate along some paths; In other words, as long as $z \geq 0$ for some of the units in each layer.
  In contrast to the ReLU, both the hyperbolic tangent and the sigmoid functions cause significant attenuation in elements of the $\boldsymbol{\delta}$ in deeper layers even when their inputs are from a small interval centered at zero (i.e., are not from the saturated regions).





Deep Learning
Deep Feed Forward Networks

- **Weight Initialization**

  Proper weight initialization ensures that, at the start of training, the magnitudes of $a^{(l)}$ and $\delta^{(l)}$ will not attenuate to zero or grow very large. With proper initialization, in the forward pass, starting from an input with zero mean and unit variance, the variance of the elements of the output of each layer remains the same as that for the layer bellow it. Likewise, in the back-propagation, the variance of the elements of the $\boldsymbol{\delta}$ at the output of each layer remains the same as that for the layer above it.

  He Initialization

  For layers with ReLU activation, the weights can be drawn from either a normal distribution $w^{(l)} \sim \mathcal{N}(0, \frac{2}{O_{l-1}})$, in which O_{l-1} denotes the size of the input to the l^{th} layer, or a uniform distribution $w^{(l)} \sim U[-\sqrt{\frac{6}{O_{l-1}}}, \sqrt{\frac{6}{O_{l-1}}}]$.

  Glorot Initialization

  For layers with hyperbolic tangent activation, the weights can be drawn from either a normal distribution $w^{(l)} \sim \mathcal{N}(0, \frac{2}{O_l + O_{l-1}})$ or a uniform distribution $w^{(l)} \sim U[-\sqrt{\frac{6}{O_l+O_{l-1}}}, \sqrt{\frac{6}{O_l+O_{l-1}}}]$.

Shattered Gradients

When the gradients of the cost w.r.t the weights and bioases of the l^{th} layer, i.e., $\frac{\partial C}{\partial \tilde{\boldsymbol{W}}^{(l)}}$, exhibit unexpected behavior, the learning algorithm is said to suffer from **shattered gradients**. This effect arises from the sequential dependance of layer outputs and also the simultaneous update of all layers. For example, from the chain rule, $\frac{\partial C}{\partial W_{11}^{(2)}} = a_1^{(1)} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \sum_{i=1}^{4} W_{i,1}^{(3)} \frac{\partial a_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial C}{\partial a_i^{(3)}}$. The infinitesimal change in $W_{11}^{(2)}$ is computed under the assumption that its constituent terms don't change as $W_{11}^{(2)}$ undergoes this change. However, this assumption doesn't hold given that $\boldsymbol{a}^{(2)}$, $\boldsymbol{a}^{(3)}$ and, ultimately, C are themselves dependent on $W_{11}^{(2)}$ and also that $W_{11}^{(2)}$ and all $W_{i,1}^{(3)}$ are updated simultaneously.

- **Shortcut Connections: The Residual Layer**

  Adding shortcut (skip) connections to layers results in multiple paths of different lengths (including the direct/identity path) between the input and the output.
  In general, gradients through shorter paths will be better behaved. Since both the identity term and various short chains of derivatives will contribute to the derivative for each layer, networks with shortcut connections suffer less from shattered gradients.

Batch Normalization

The learning algorithm can also greatly benefit from learning the moments of distribution (i.e., the mean and variance) of the elements of the input to each layer alongside the weighs and biases of that layer.
The batch normalization (BN) procedure, in the first step, standardizes each element of the layer's input with the mean and variance that is computed over the current batch for that element and, in the second step, scales and shifts the standardized elements with learned parameters.
In the training phase, for $a_k^{(l-1)}$, the BN procedure computes $\mu_{k,B}$ and $\sigma_{k,B}^2$ from the entire batch and then transforms it to $a_k^{''(l-1)} = \gamma_k a_k^{'(l-1)} + \beta_k$ in which $a_k^{'(l-1)} = \frac{1}{\sigma_{k,B}}(a_k^{(l-1)} - \mu_{k,B})$.
In the inference phase, $a_k^{'(l-1)}$ is computed using the average of the values computed for $\mu_{k,B}$ and $\sigma_{k,B}^2$ during the training.

Deep Learning
Experiment Setup in TensorFlow 2

Training

```
...
import tensorflow as tf
...

(1) seq_network_1=get_seq_network_1(arg_1, ...)

(2) model        =ml_algorithm(seq_network_1=seq_network_1,
                               ...                          )

model.compile(optimizer =tf.keras.optimizers.Adam(...),
              loss      =...                             ,
              <metrics> =...                             )

(3) callbacks=[callback_custom_checkpoint    ()   ,
               callback_custom_monitor       (...),
               callback_custom_history       (...),
               tf.keras.callbacks.Tensorboard(...) ]

(4) save_data_to_tfrecords_format (arg_1,...)

(5) data_train     =read_tfrecords(mode='train'     ,...)
data_validation=read_tfrecords(mode='validation',...)

model.fit(x              =data_train     ,
          validation_data=data_validation,
          callbacks      =callbacks      ,
          ...                            )
```

Inference

```
...
import tensorflow as tf
...
model      =tf.keras.models.load_model(path_to_checkpoint,
                                       ...                 )
data       = ...
predictions=model(data)
...
```

(2)

```
...
import tensorflow as tf
...

class ml_algorithm(tf.keras.Model):

    def __init__(self,seq_network_1,...):
     super().__init__()
     self.network_1=seq_network_1
     ...

    def compile(self,optimizer,loss,<metrics>):
     super().compile()
     self.optimizer=optimizer
     self.loss     =loss
     self.<metrics>=<metrics>

    def call(self,X,training=None):
     x=self.<network>(X,training=training)
     ...
     return ...

    def train_step(self,data):
     if isinstance(data,tuple):
      x,y=data
     ...
     with tf.GradientTape() as tape:
      predictions=self     (x,training=True)
      ...
      loss       =self.loss(y,predictions  )
     gradients   =tape.gradient    (    loss     , self.<network>.trainable_weights )
     self.optimizer.apply_gradients(zip(gradients, self.<network>.trainable_weights))
     return {'loss': loss,
             ...          }
```

(1)

```
...
import tensorflow as tf
...

def get_seq_network(arg_1,...):
 X    =tf.keras.Input(shape=..., ...)
 ...
 x    =tf.keras.layers.Dense             (...)(X)
 x    =tf.keras.layers.BatchNormalization(...)(x)
 x    =tf.keras.layers.ReLU              (...)(x)
 ...
 return tf.keras.Model(inputs=X, outputs=x, name=...)
```

(3)

```
...
import tensorflow as tf
...

class callback_custom_checkpoint(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs=None):
        tf.keras.models.save_model(self.model ,path_to_checkpoint)
```

Deep Learning
Experiment Setup in TensorFlow 2

④

```
...
import tensorflow as tf
...

def save_data_to_tfrecords_format(...):

 def serialize_example(x,y):
  feature      ={'x': tf.train.Feature(bytes_list=tf.train.BytesList(value=[x.tostring()])),
                 'y': tf.train.Feature(int64_list=tf.train.Int64List(value=[y          ]))
  example_proto=tf.train.Example(features=tf.train.Features(feature=feature))
  return example_proto.SerializeToString()

 def write_serialized_example(x,y,mode):
  filename =...
  with tf.io.TFRecordWriter(filename) as writer:
   for i in range(number_of_examples):
    example=serialize_example(x[i],y[i])
    writer.write(example)

 def get_train_validation_splits(x,y,...):
  ...
  return x_train, y_train, x_validation, y_validation

x,y                                  =get_data                    (...    )
x_train, y_train, x_validation, y_validation=get_train_validation_splits(x,y,...)
write_serialized_example(x_train     , y_train     ,'train'       )
write_serialized_example(x_validation, y_validation,''validation'')
```

⑤

```
...
import tensorflow as tf
...

def read_tfrecords(mode, ...):
 ...
 feature ={'x': tf.io.FixedLenFeature([],tf.string),
           'y': tf.io.FixedLenFeature([],tf.int64 ) }
 def parse_function(example_proto):
  parsed_example=tf.io.parse_single_example(example_proto,feature         )
  x             =tf.io.decode_raw          (parsed_example['x'],tf.float32)
  x.set_shape(shape_of_x)
  y             =parsed_example['y']
  ...
  return x,y
 dataset =tf.data.TFRecordDataset(filenames                                    )
 dataset =dataset.map            (parse_function                               )
 dataset =dataset.batch          (batch_size     , drop_remainder          =True)
 dataset =dataset.shuffle        (buffer_capacity, reshuffle_each_iteration=True)
 return dataset
```

Appendices
A Medley of Topics

The Bias-Variance Decomposition

$$E\left[\left(y - f_D\left(\boldsymbol{x}\right)\right)^2\right] = E\left[\left(\bar{y}\left(\boldsymbol{x}\right) - \bar{f_D}\left(\boldsymbol{x}\right)\right)^2\right] + E\left[\left(f_D\left(\boldsymbol{x}\right) - \bar{f_D}\left(\boldsymbol{x}\right)\right)^2\right] + E\left[\left(y - \bar{y}\left(\boldsymbol{x}\right)\right)^2\right]$$

The **bias**, i.e., $E\left[\left(\bar{y}\left(\boldsymbol{x}\right) - \bar{f_D}\left(\boldsymbol{x}\right)\right)^2\right]$, captures the model's inherent error (i.e., that portion of the error that can't be reduced by increasing the size of the training set). The **variance**, i.e., $E\left[\left(f_D\left(\boldsymbol{x}\right) - \bar{f_D}\left(\boldsymbol{x}\right)\right)^2\right]$, captures how much the model changes if trained on a different training set. In other words, it answers the question: How over-specialized is the model to a particular training set? The **noise**, i.e., $E\left[\left(y - \bar{y}\left(\boldsymbol{x}\right)\right)^2\right]$, captures the ambiguity due to the data-generating distribution of the training set and feature representation.

Bayesian Inference

$$E\left[\hat{y}\right] = \boldsymbol{y}^T \tilde{\boldsymbol{X}} \left(\tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}} + \sigma^2 \left(\boldsymbol{\Sigma}^{\text{prior}}\right)^{-1}\right)^{-1} \tilde{\boldsymbol{x}}$$

$$\text{Var}\left[\hat{y}\right] = \sigma^2 \tilde{\boldsymbol{x}}^T \left(\tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}} + \sigma^2 \left(\boldsymbol{\Sigma}^{\text{prior}}\right)^{-1}\right)^{-1} \tilde{\boldsymbol{x}}$$